PHASE1-1.1 PART 2: Cost Agent Skeleton - Execution & Validation

OptiInfra Development Series

Phase: Cost Agent (Week 2-3)

Component: Cost Agent Foundation Validation

Estimated Time: 15 minutes validation

Dependencies: PHASE1-1.1 PART 1 (Already complete from P-03)

📋 Overview

This document provides comprehensive validation steps to ensure the Cost Agent Skeleton (completed in P-03) is ready for Phase 1 development.

✅ Pre-Validation Checklist

Before starting validation, ensure:

- [ ] All services from Foundation phase are running
- [ ] PostgreSQL, ClickHouse, Qdrant, Redis are healthy
- [ ] Orchestrator is running on port 8080
- [ ] Cost Agent source code exists in `services/cost-agent/`
- [ ] Python 3.11+ is installed
- [ ] Dependencies are installed (`pip install -r requirements.txt`)

🚀 Step-by-Step Validation

Step 1: Start the Cost Agent

```bash
```

```
# Navigate to cost agent directory
cd ~/optiinfra/services/cost-agent

# Create virtual environment (if not exists)
python3 -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt

# Start the service
python -m src.main
```

Expected Output:

```
INFO:     Will watch for changes in these directories: ['/home/user/optiinfra/services/cost-agent']
INFO:     Uvicorn running on http://0.0.0.0:8001 (Press CTRL+C to quit)
INFO:     Started reloader process [12345] using StatReload
INFO:     Started server process [12346]
INFO:     Waiting for application startup.
INFO:     Starting Cost Agent...
INFO:     Database connection successful
INFO:     Application startup complete.
```

✅ **Success Criteria:**

- Service starts without errors
- Port 8001 is listening
- Database connection successful message appears

Step 2: Test Health Endpoints

2.1 Main Health Check

```
bash

curl http://localhost:8001/api/v1/health | jq
```

Expected Response:

```json
{
  "status": "healthy",
  "timestamp": "2025-10-21T10:30:45.123456",
  "version": "1.0.0",
  "database": {
    "postgres": "healthy",
    "clickhouse": "healthy",
    "qdrant": "healthy",
    "redis": "healthy"
  }
}
```

✅ **Success Criteria:**

- HTTP 200 status code
- `status` field is "healthy"
- All 4 databases report "healthy"

2.2 Readiness Probe

```bash
curl http://localhost:8001/api/v1/ready
```

Expected Response:

```json
{
  "status": "ready"
}
```

✅ **Success Criteria:**

- HTTP 200 status code
- Service is ready to receive traffic

2.3 Liveness Probe

```bash
```

```
curl http://localhost:8001/api/v1/live
```

Expected Response:

```
json

{
  "status": "alive"
}
```

✅ **Success Criteria:**

- HTTP 200 status code
- Service is alive and responding

Step 3: Test Prometheus Metrics

```
bash

# Fetch metrics endpoint
curl http://localhost:8001/metrics

# Check for cost agent specific metrics
curl http://localhost:8001/metrics | grep cost_savings_total
curl http://localhost:8001/metrics | grep cost_recommendations_total
curl http://localhost:8001/metrics | grep spot_migration_success_rate
```

Expected Output:

```
# HELP cost_savings_total Total cost savings in USD
# TYPE cost_savings_total counter
cost_savings_total{provider="aws",optimization_type="spot"} 0.0

# HELP cost_recommendations_total Total number of cost recommendations
# TYPE cost_recommendations_total counter
cost_recommendations_total{type="spot",confidence="high"} 0.0

# HELP spot_migration_success_rate Success rate of spot instance migrations
# TYPE spot_migration_success_rate gauge
spot_migration_success_rate 0.0
```

✅ **Success Criteria:**

- Metrics endpoint returns Prometheus format
- Cost-specific metrics are present
- Base metrics (requests_total, errors_total) are present

Step 4: Test FastAPI Documentation

```
bash

# Open in browser or curl
curl http://localhost:8001/docs
```

Or open in browser:

- Navigate to: http://localhost:8001/docs
- Should see interactive API documentation (Swagger UI)

✅ **Success Criteria:**

- Swagger UI loads successfully
- See endpoints: `/api/v1/health`, `/api/v1/ready`, `/api/v1/live`
- See analyze endpoints from P-04
- Can test endpoints interactively

Step 5: Test Basic Workflow (from P-04)

```
bash

# Test analyze endpoint
curl -X POST http://localhost:8001/api/v1/analyze \
  -H "Content-Type: application/json" \
  -d '{
    "customer_id": "test-customer",
    "time_period_days": 30
  }' | jq
```

Expected Response:

json


```
{
  "analysis_id": "uuid-string",
  "customer_id": "test-customer",
  "status": "completed",
  "analysis": {
    "total_cost": 120000,
    "waste_identified": 60000,
    "waste_percentage": 50.0
  },
  "recommendations": [
    {
      "type": "spot_migration",
      "description": "Migrate 10 instances to spot",
      "estimated_savings": 18000,
      "confidence": 0.85
    }
  ],
  "summary": "Found 50% cost waste...",
  "timestamp": "2025-10-21T10:30:45.123456"
}
```


✅ **Success Criteria:**

- HTTP 200 status code
- Analysis completes successfully
- Returns analysis, recommendations, and summary
- Workflow nodes execute in order

Step 6: Test Spot Migration Workflow (from P-05)

bash

```
# Test spot migration endpoint
curl -X POST http://localhost:8001/api/v1/spot-migration \
  -H "Content-Type: application/json" \
  -d '{
    "customer_id": "test-customer",
    "instance_ids": ["i-1234567890abcdef0"],
    "rollout_strategy": "gradual"
  }' | jq
```

Expected Response:

```
json

{
  "migration_id": "uuid-string",
  "customer_id": "test-customer",
  "status": "completed",
  "phases": {
    "analysis": "completed",
    "coordination": "completed",
    "execution": "completed",
    "monitoring": "completed"
  },
  "results": {
    "instances_migrated": 1,
    "estimated_savings_monthly": 2500,
    "quality_score": 0.98,
    "rollback_triggered": false
  },
  "timestamp": "2025-10-21T10:31:15.123456"
}
```

✅ **Success Criteria:**

- HTTP 200 status code
- All 4 workflow phases complete
- Returns savings estimate
- Quality score above threshold (>0.95)

Step 7: Verify Database Connections

7.1 PostgreSQL

```bash
# Check cost agent can query PostgreSQL
psql $DATABASE_URL -c "SELECT COUNT(*) FROM customers;"
```

Expected:

```
 count
-------
     0
(1 row)
```

7.2 ClickHouse

```bash
# Check ClickHouse connection
curl "http://localhost:8123/?query=SELECT%201"
```

Expected:

```
1
```

7.3 Qdrant

```bash
# Check Qdrant collections
curl http://localhost:6333/collections | jq
```

Expected:

```json
```

```
{
  "result": {
    "collections": [
      {"name": "agent_interactions"},
      {"name": "optimization_patterns"},
      {"name": "recommendation_embeddings"}
    ]
  }
}
```

7.4 Redis

```
bash

# Check Redis connection
redis-cli ping
```

Expected:

```
PONG
```

✅ **Success Criteria:**

- All 4 databases respond successfully
- Cost agent can connect to each database
- No connection errors in logs

Step 8: Test Prometheus Integration

```
bash

# Check if Prometheus is scraping cost agent
curl http://localhost:9090/api/v1/targets | jq '.data.activeTargets[] | select(.labels.job=="cost-agent")'
```

Expected Response:

```
json
```

```
{
  "discoveredLabels": {
    "__address__": "cost-agent:8001",
    "job": "cost-agent"
  },
  "labels": {
    "instance": "cost-agent:8001",
    "job": "cost-agent"
  },
  "scrapePool": "cost-agent",
  "scrapeUrl": "http://cost-agent:8001/metrics",
  "lastError": "",
  "lastScrape": "2025-10-21T10:30:45.123Z",
  "lastScrapeDuration": 0.012,
  "health": "up"
}
```


✅ **Success Criteria:**

- Target health is "up"
- Last scrape successful
- No scrape errors

Step 9: Verify Grafana Dashboard

```bash
# Open Grafana Cost Agent dashboard
open http://localhost:3000/d/optiinfra-cost
```

Or via API:

```bash
curl -u admin:optiinfra_admin \
  http://localhost:3000/api/dashboards/uid/optiinfra-cost | jq '.dashboard.title'
```

Expected:

```
"Cost Agent"
```

✅ **Success Criteria:**

- Dashboard loads in Grafana
- All panels render (may show 0 values initially)
- No data source errors

Step 10: Run Automated Tests

```bash
# Run all tests
cd ~/optiinfra/services/cost-agent
pytest tests/ -v

# Run with coverage
pytest tests/ -v --cov=src --cov-report=term-missing
```

Expected Output:

```
tests/test_health.py::test_health_check PASSED
tests/test_health.py::test_ready_check PASSED
tests/test_health.py::test_live_check PASSED
tests/test_workflow.py::test_analyze_workflow PASSED
tests/test_workflow.py::test_workflow_nodes PASSED
tests/test_analyze_api.py::test_analyze_endpoint PASSED
tests/test_analyze_api.py::test_analyze_invalid_input PASSED

---------- coverage: platform linux, python 3.11.5 -----------
Name                              Stmts   Miss  Cover   Missing
-----------------------------------------------------------------
src/__init__.py                       0      0   100%
src/main.py                          45      2    96%   78-79
src/config.py                        35      0   100%
src/api/health.py                    42      1    98%   65
src/api/analyze.py                   28      0   100%
src/workflows/cost_optimization.py   35      1    97%   82
src/nodes/analyze.py                 22      0   100%
src/nodes/recommend.py               18      0   100%
src/nodes/summarize.py               15      0   100%
src/metrics.py                       45      2    96%   92-93
-----------------------------------------------------------------
TOTAL                               285      6    98%

====== 7 passed in 2.34s ======
```

✅ **Success Criteria:**

- All tests pass (7/7 or more)
- Code coverage ≥80% (achieved: 98%)
- No test failures or errors

Step 11: Load Testing (Optional)

```
bash
```

```
# Install load testing tool
pip install locust

# Create locustfile.py
cat > locustfile.py << 'EOF'
from locust import HttpUser, task, between

class CostAgentUser(HttpUser):
    wait_time = between(1, 3)

    @task(3)
    def health_check(self):
        self.client.get("/api/v1/health")

    @task(1)
    def analyze(self):
        self.client.post("/api/v1/analyze", json={
            "customer_id": "load-test",
            "time_period_days": 30
        })
EOF

# Run load test
locust -f locustfile.py --host=http://localhost:8001 --users=10 --spawn-rate=2 --run-time=60s --headless
```

Expected Output:

```
Type     Name                          # reqs      # fails |    Avg     Min     Max  Median |   req/s failures/s
--------|-------------------------------|-------|-------------|-------|-------|-------|-------|--------|--------
GET      /api/v1/health                  450       0     |     12       8      45      11 |    7.50        0.00
POST     /api/v1/analyze                 150       0     |    125      98     245     120 |    2.50        0.00
--------|-------------------------------|-------|-------------|-------|-------|-------|-------|--------|--------
         Aggregated                      600       0     |     42       8     245      15 |   10.00        0.00

Response time percentiles (approximated)
Type     Name                                50%    66%    75%    80%    90%    95%    98%    99%  99.9% 99.99%   100% #
reqs
--------|-------------------------------------|--------|------|------|------|------|------|------|------|------|------|------
GET      /api/v1/health                       11     12     13     14     16     19     25     35     45     45     45    450
POST     /api/v1/analyze                     120    130    145    155    180    200    220    235    245    245    245    150
--------|-------------------------------------|--------|------|------|------|------|------|------|------|------|------|------
         Aggregated                          15     20     30     50    150    180    210    230    245    245    245    600
```

✅ **Success Criteria:**

- 0% failure rate
- P95 latency <200ms for health checks
- P95 latency <2s for analyze endpoint
- Service handles 10 concurrent users

🐛 Troubleshooting

Issue 1: Service Won't Start

Symptoms:

```
ERROR:    [Errno 48] Address already in use
```

Solution:

```bash
# Check what's using port 8001
lsof -i :8001

# Kill the process
kill -9 <PID>

# Or use a different port
uvicorn src.main:app --host 0.0.0.0 --port 8002
```

Issue 2: Database Connection Failures

Symptoms:

```
ERROR: Database connection failed: could not connect to server
```

Solution:

```bash
```

```
# Check if databases are running
docker ps | grep -E "postgres|clickhouse|qdrant|redis"

# Start databases if not running
cd ~/optiinfra
docker-compose up -d postgres clickhouse qdrant redis

# Wait 10 seconds and retry
sleep 10
curl http://localhost:8001/api/v1/health
```

Issue 3: Import Errors

Symptoms:

```
ModuleNotFoundError: No module named 'shared'
```

Solution:

```
bash

# Ensure shared utilities are in PYTHONPATH
export PYTHONPATH=$PYTHONPATH:~/optiinfra

# Or install in development mode
cd ~/optiinfra
pip install -e .

# Restart the service
python -m src.main
```

Issue 4: Metrics Not Appearing

Symptoms:

- Metrics endpoint returns empty or missing metrics

Solution:

```
bash
```

```
# Check if prometheus_client is installed
pip list | grep prometheus-client

# Trigger some requests to generate metrics
curl http://localhost:8001/api/v1/health
curl http://localhost:8001/api/v1/health
curl http://localhost:8001/api/v1/health

# Check metrics again
curl http://localhost:8001/metrics | grep requests_total
```

Issue 5: Tests Failing

Symptoms:

```
FAILED tests/test_health.py::test_health_check
```

Solution:

```
bash

# Run with verbose output
pytest tests/test_health.py -vv

# Check test logs
pytest tests/test_health.py -v --log-cli-level=DEBUG

# Ensure databases are running
docker-compose up -d postgres clickhouse qdrant redis

# Retry tests
pytest tests/test_health.py -v
```

📊 Success Metrics Summary

After completing all validation steps, verify these metrics:

Metric	Target	Status
Service Starts	Without errors	⬜
Health Endpoint	Returns 200 OK	⬜
All Databases	Report "healthy"	⬜
Metrics Endpoint	Exposes Prometheus metrics	⬜
FastAPI Docs	Accessible at /docs	⬜
Basic Workflow	Completes successfully	⬜
Spot Workflow	Completes successfully	⬜
Prometheus Scraping	Target is "up"	⬜
Grafana Dashboard	Loads without errors	⬜
All Tests	Pass (7/7 minimum)	⬜
Code Coverage	≥80% (achieved: 98%)	⬜
Load Test	0% failure rate	⬜

📋 Validation Completion Checklist

Mark each item when verified:

Basic Functionality

- [ ] Service starts on port 8001
- [ ] Health check returns "healthy"
- [ ] Readiness probe returns "ready"
- [ ] Liveness probe returns "alive"
- [ ] Metrics endpoint working
- [ ] FastAPI docs accessible

Database Connectivity

- [ ] PostgreSQL connection healthy
- [ ] ClickHouse connection healthy
- [ ] Qdrant connection healthy
- [ ] Redis connection healthy

Workflows

- [ ] Basic analysis workflow works (P-04)
- [ ] Spot migration workflow works (P-05)

- [ ] Workflows complete all phases
- [ ] Results include savings estimates

Monitoring

- [ ] Prometheus scraping successfully
- [ ] All cost metrics present
- [ ] Grafana dashboard loads
- [ ] Dashboard panels show data

Testing

- [ ] All unit tests pass
- [ ] Code coverage ≥80%
- [ ] Integration tests pass
- [ ] Load test succeeds (optional)

Integration

- [ ] Integrates with orchestrator
- [ ] Uses shared utilities
- [ ] Follows project structure
- [ ] Logs to correct format

🎯 Decision Gate: Ready for Phase 1?

Answer these questions:

1. **Does the service start without errors?**
   - [ ] YES ✅ → Continue
   - [ ] NO ❌ → Debug startup issues
2. **Are all databases healthy?**
   - [ ] YES ✅ → Continue
   - [ ] NO ❌ → Fix database connections
3. **Do both workflows (P-04 and P-05) work?**
   - [ ] YES ✅ → Continue
   - [ ] NO ❌ → Review workflow implementation
4. **Are metrics being collected?**
   - [ ] YES ✅ → Continue

- [ ] NO ❌ → Fix metrics setup

5. **Do all tests pass?**
   - [ ] YES ✅ → Continue
   - [ ] NO ❌ → Fix failing tests

If all answers are YES: ✅ **APPROVED** - Ready to proceed to PROMPT 1.2 (AWS Cost Collector)

If any answer is NO: ⚠️ **BLOCKED** - Resolve issues before continuing

---

🚀 Next Steps

Once validation is complete:

Option 1: Continue to Next Prompt

"Generate PROMPT 1.2: AWS Cost Collector"

Option 2: Review Week 2 Plan

"Show me all Week 2 prompts"

Option 3: Get Help

"I'm having issues with validation step X"

📁 Validation Artifacts

Save these for your records:

1. Health Check Response

bash

```
curl http://localhost:8001/api/v1/health | jq > validation/health-check.json
```

2. Metrics Snapshot

bash

```
curl http://localhost:8001/metrics > validation/metrics-snapshot.txt
```

3. Test Results

```
bash

pytest tests/ -v --html=validation/test-report.html --self-contained-html
```

4. Load Test Results

```
bash

locust -f locustfile.py --host=http://localhost:8001 --users=10 --run-time=60s --html=validation/load-test.html
```

📚 Additional Resources

- FastAPI Documentation
- LangGraph Documentation
- Prometheus Python Client
- pytest Documentation

✅ Completion Sign-off

Validated By: _____

Date: _____

Status: ☐ PASS / ☐ FAIL

Notes: _____

Document Version: 1.0

Status: ✅ Validation Guide Complete

Last Updated: October 21, 2025

Previous: PHASE1-1.1 PART 1 (Code)

Next: PHASE1-1.2 (AWS Cost Collector)